No Time to Do It All!

Approaching Overload on DevOps Teams

MAR 26
2025

Alex Wise
@aws-snarkitect@bluesky.social
contact@alexwise.guru

@aws-snarkitect@bsky.social

- SWE focused on reliability
- I believe in open source software
- ← That's my dog!

Overload: The flow of work into the system is greater than the rate of work it can perform.


Bee-in’
Busy

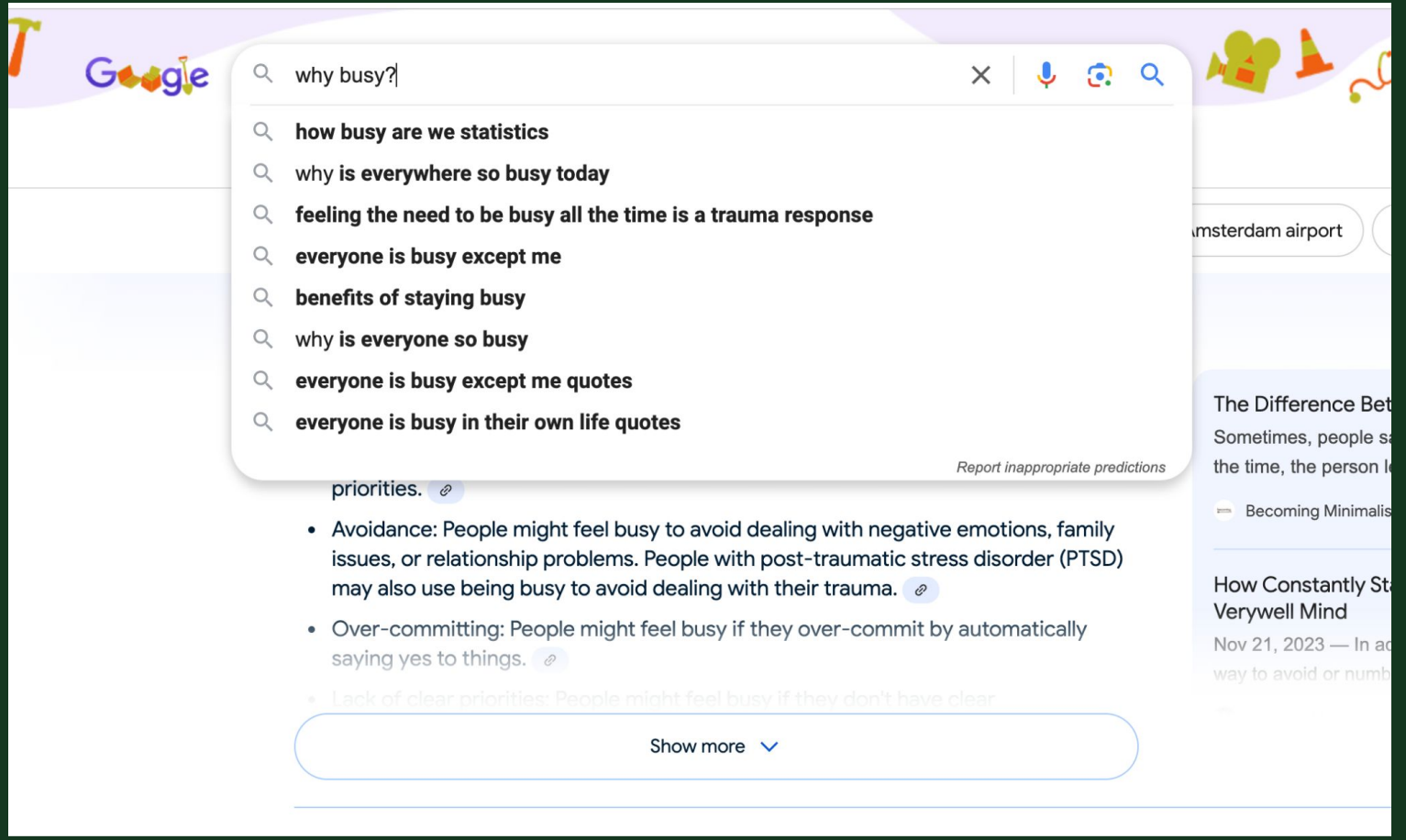
CrowdStrike ex-employees: 'Quality control was not part of our process'

Some former employees said quality checks on software were rushed at times to get products launched quickly.

"It was hard to get people to do sufficient testing sometimes," said Preston Sego, who worked at CrowdStrike from 2019 to 2023. His job was to review the tests completed by user experience developers that alerted engineers to bugs before proposed coding changes were released to customers. Sego said he was fired in February 2023 as an "insider threat" after he criticized the company's return-to-work policy on an internal Slack channel. That's

https://www.semafor.com/article/09/12/2024/ex-crowdstrike-employees-detail-rising-technical-errors-before-july-outage

Wayfinder™


Google
why busy?
how busy are we statistics
why is everywhere so busy today
feeling the need to be busy all the time is a trauma response
everyone is busy except me
benefits of staying busy
why is everyone so busy
everyone is busy except me quotes
everyone is busy in their own life quotes
Report inappropriate predictions
amsterdam airport
priorities.
Avoidance: People might feel busy to avoid dealing with negative emotions, family issues, or relationship problems. People with post-traumatic stress disorder (PTSD) may also use being busy to avoid dealing with their trauma.
Over-committing: People might feel busy if they over-commit by automatically saying yes to things.
Show more
The Difference Bet
Sometimes, people s
the time, the person l
Becoming Minimalis
How Constantly St
Verywell Mind
Nov 21, 2023 — In ac
way to avoid or numb

Search Donate Fundraise gofundme About Alex

Alex is Climbing a mountain to think because he feels busy.

Alex Wise is organizing this fundraiser.

$5,000 goal

Share

Donate now

This fundraiser is located near you

Become the first supporter
Your donation matters

GoFundMe protects your donation

We guarantee you a full refund for up to a year in the rare case that fraud occurs. See our GoFundMe Giving Guarantee.

So I came down from the mountain thinking I had uncovered two fundamental truths.

The paper introduces the theory of graceful extensibility which expresses fundamental characteristics of the adaptive universe that constrain the search for sustained adaptability. The theory explains the contrast between successful and unsuccessful cases of sustained adaptability for systems that serve human purposes. Sustained adaptability refers to the ability to continue to adapt to changing environments, stakeholders, demands, contexts, and constraints (in effect, to adapt how the system in question adapts). The key new concept at the heart of the theory is graceful extensibility. Graceful extensibility is the opposite of brittleness, where brittleness is a sudden collapse or failure when events push the system up to and beyond its boundaries for handling changing disturbances and variations. As the opposite of brittleness, graceful extensibility is the ability of a system to extend its capacity to adapt when surprise events challenge its boundaries. The theory is presented in the form of a set of 10 proto-theorems derived from just two assumptions—in the adaptive universe, **resources are finite** and **change continues** The theory contains three subsets of fundamentals: managing the risk of saturation, networks of adaptive units, and outmaneuvering constraints. The theory attempts to provide a formal base and common language that characterizes how complex systems sustain and fail to sustain adaptability as demands change.

"The theory of graceful extensibility: basic rules that govern adaptive systems" Woods, 2018

- ~~“Real Engineers would never do that.”~~
- ~~“They made their bed and are sleeping in it”~~
- ~~“They are reaping what they”ve sowed.”~~
- ~~Greedy business intentionally denied critical reinvestment.~~

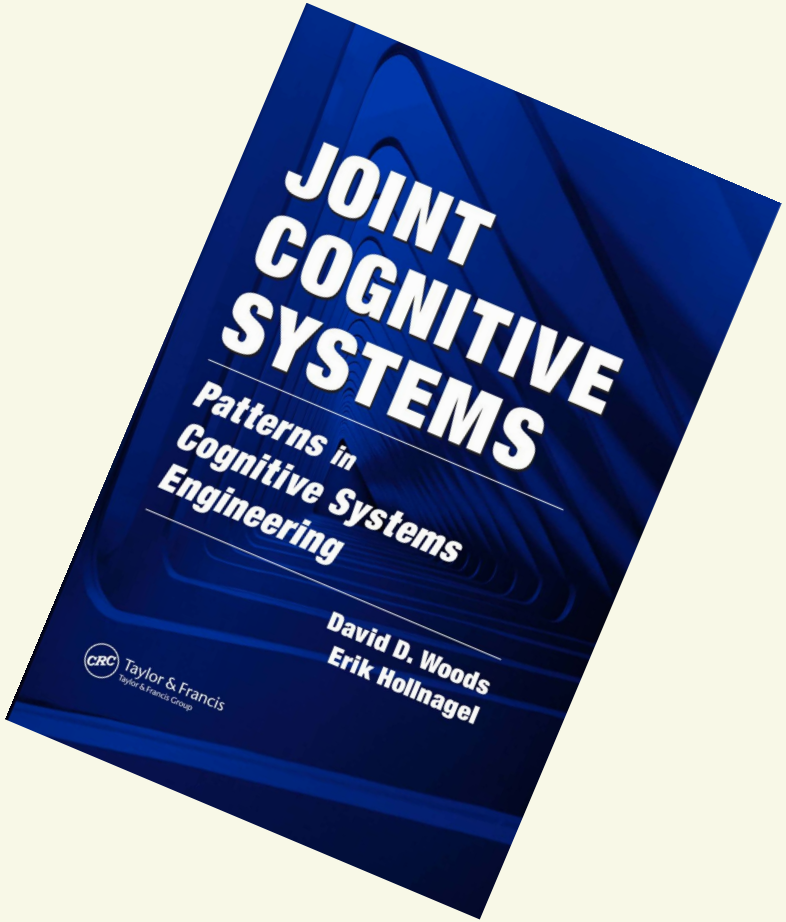
Feeling overload, making hard decisions in the face of uncertainty, and worrying about how you're going to pay down cognitive debt is just a natural part of any system, and everyone in this room should **have empathy** and reserve judgment for teams feeling this pain.

This is also why it’s important to learn to be **efficient** when faced with these resource pressures, and to embrace change as it comes.

Wayfinding

“Joint Cognitive Systems”, Woods and Hollnagel

- ❌ Shed Load
- Reduce Thoroughness
- ✅ Shift Work in Time
- Recruit Resources



If you hear these on your team, it could be a response to overload.

Shedding Load

“Does anyone know why this never got done?”
“Did anyone fix ____?”
“Did we ever circle back with X about this?”
“I think we were supposed to do that last quarter.”

Reducing Thoroughness

“Oh, this code doesn’t have any tests.”
“Did this change not get committed?”
“Where’s the documentation for this?”

It’s also important to celebrate the times we handle overload in a more positive way.

Shift Work in Time

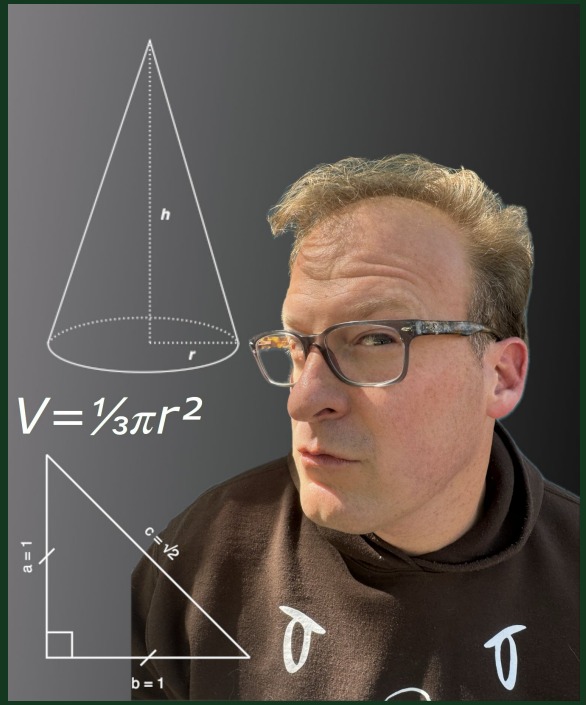
“Hey good work, we saw we weren’t going to be able to accommodate this right now so we got it shifted in time to next quarter.”

Recruit Resources

“We’re a little busy with other priority work right now, is it possible to have X team handle this part?”

Wayfinding

So I thought I'd apply some engineering...



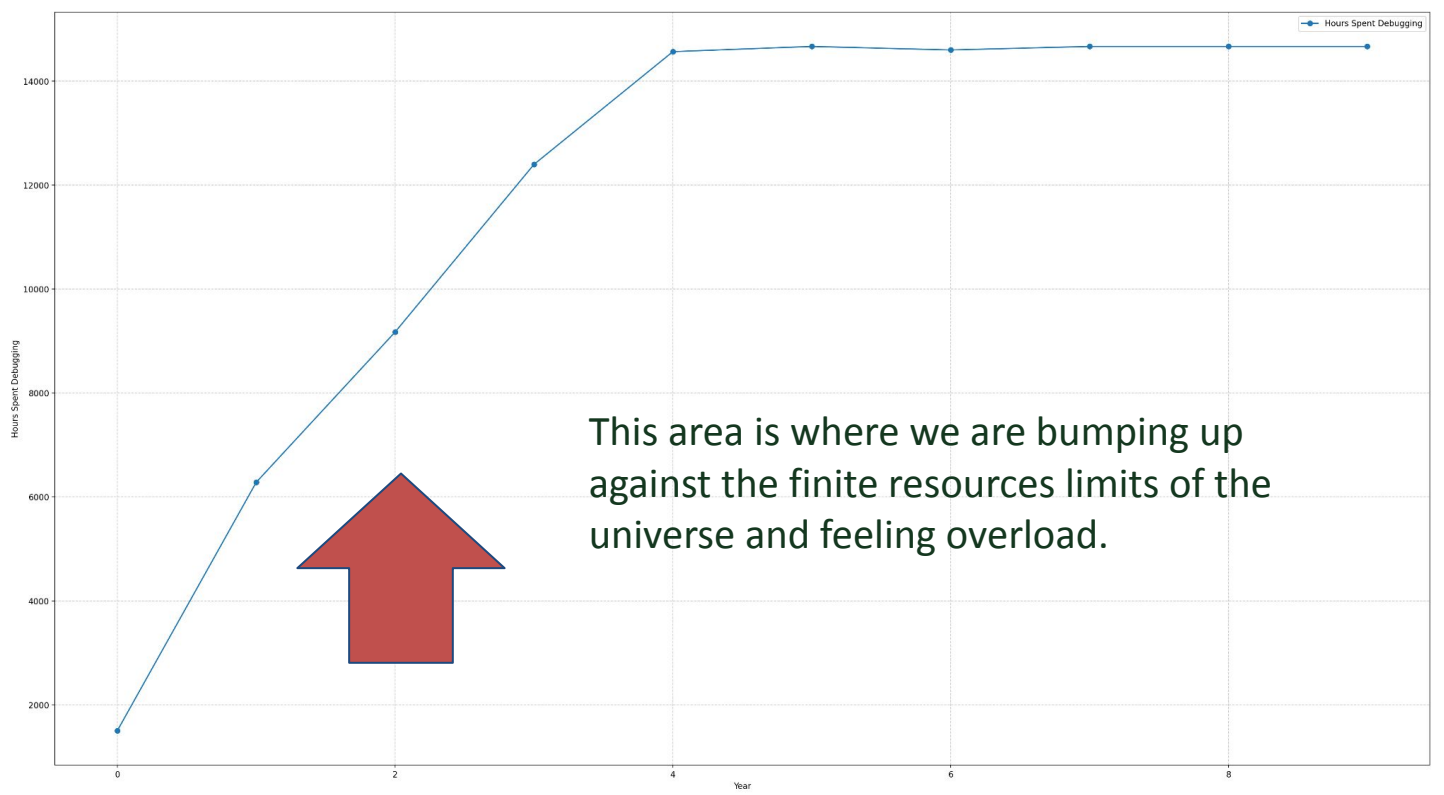
Start with Some Assumptions

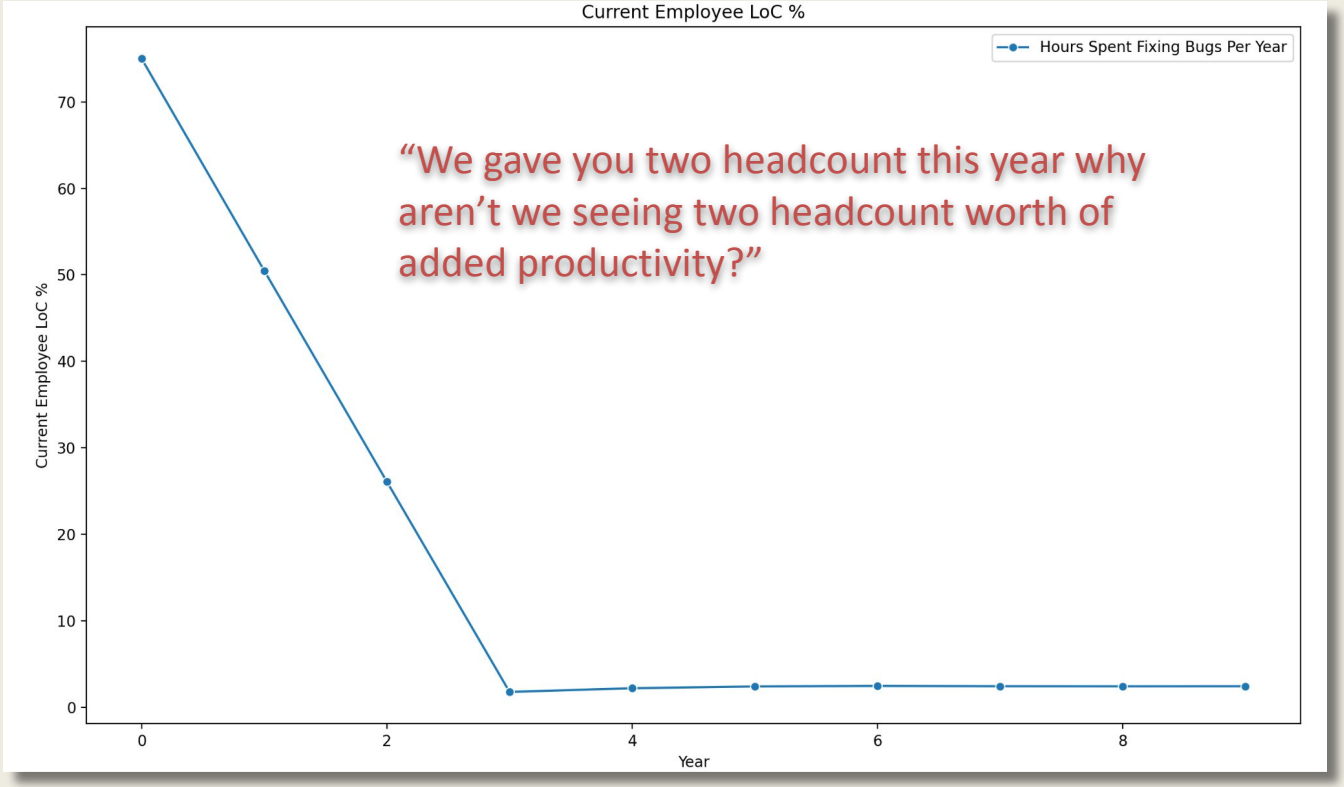
- Single Service, Greenfield, 4 SWE team
- Starts at 150k LoC written in first year
- No additional code written after that.
- Every year, the employee with the longest tenure is replaced

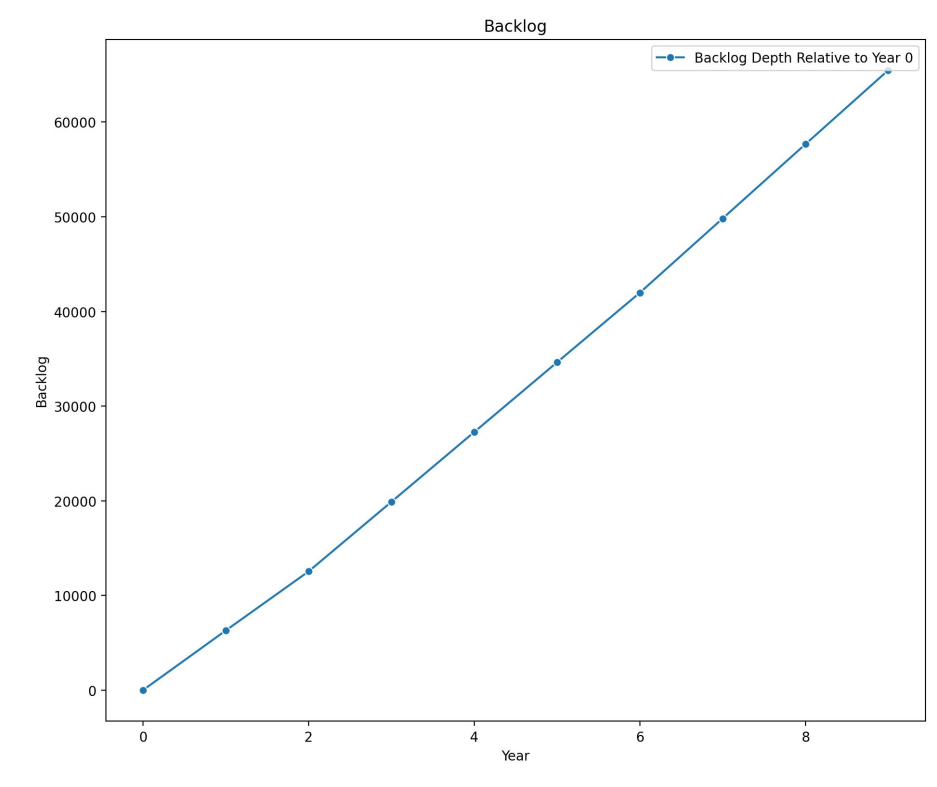
More Assumptions 🥱😴

- Each year, bugs are found for every 20k LoC, randomly distributed throughout the codebase.
- If the author of that line is still at the company, it takes 1 hour to fix
- If the author is not, it takes 10 hours to fix.
- We want to measure hours spent fixing bugs

So what does that look like?

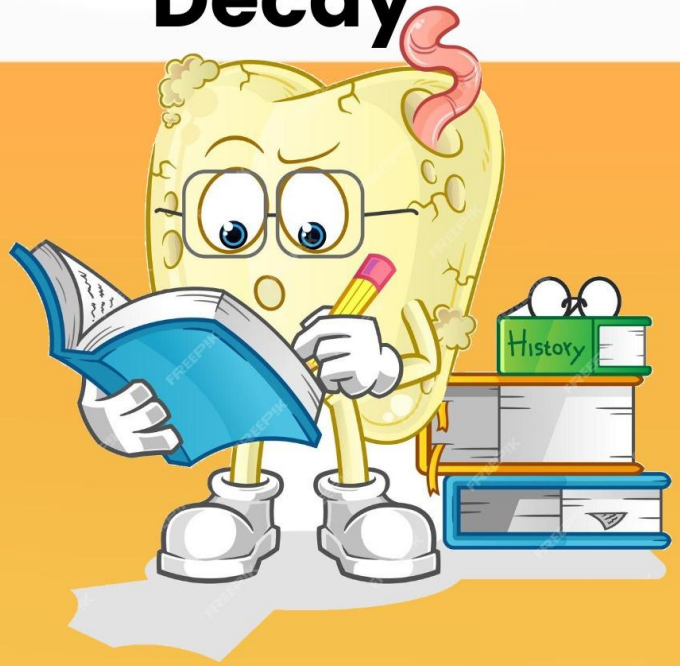



Current Employee LoC %
Hours Spent Fixing Bugs Per Year
"We gave you two headcount this year why aren't we seeing two headcount worth of added productivity?"
Current Employee LoC %
0
10
20
30
40
50
60
70
Year
0
2
4
6
8


Backlog
Backlog Depth Relative to Year 0
60000
50000
40000
30000
20000
10000
0
Backlog
0
2
4
6
8
Year

THIS IS

Knowledge Decay

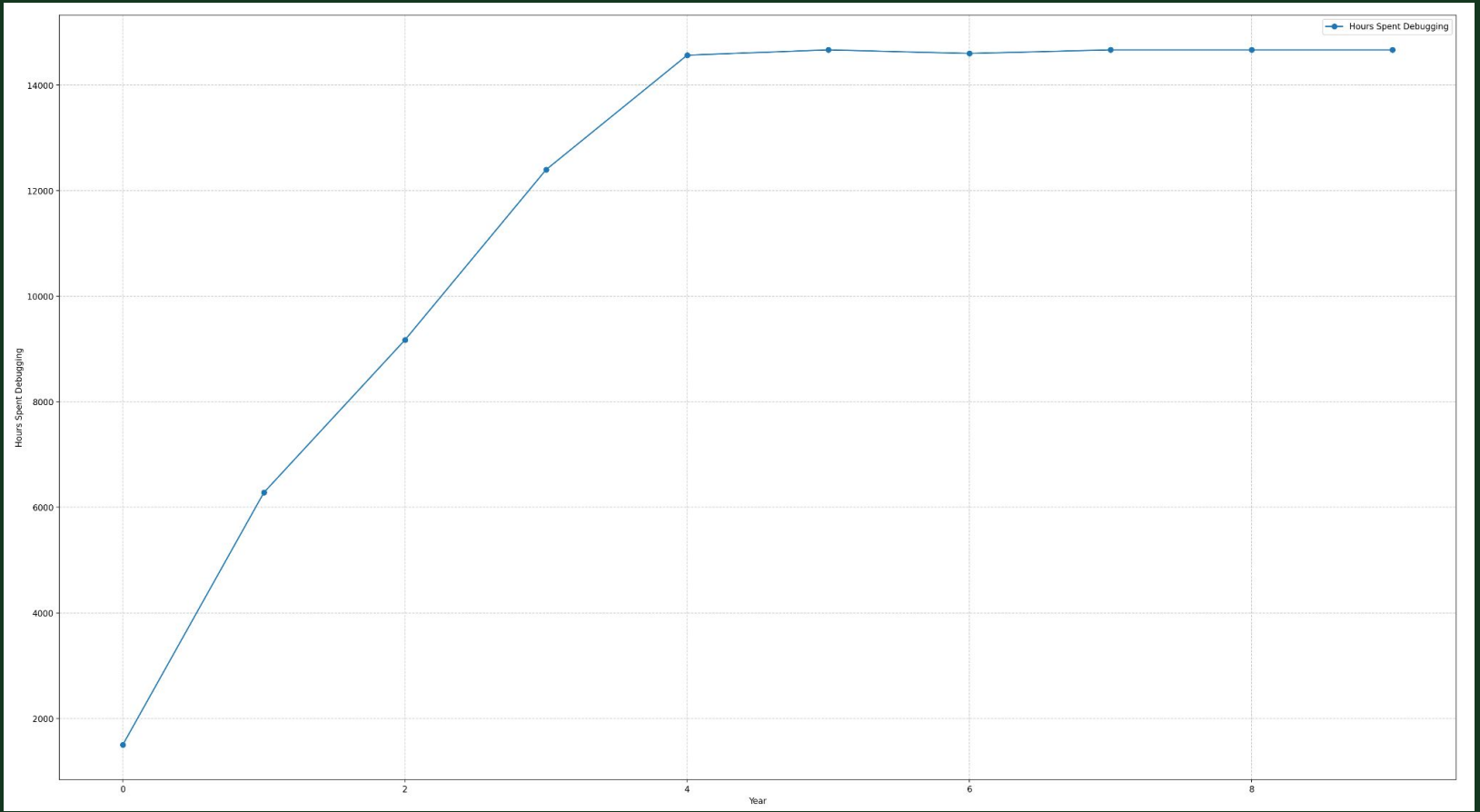


But what can we do about it?

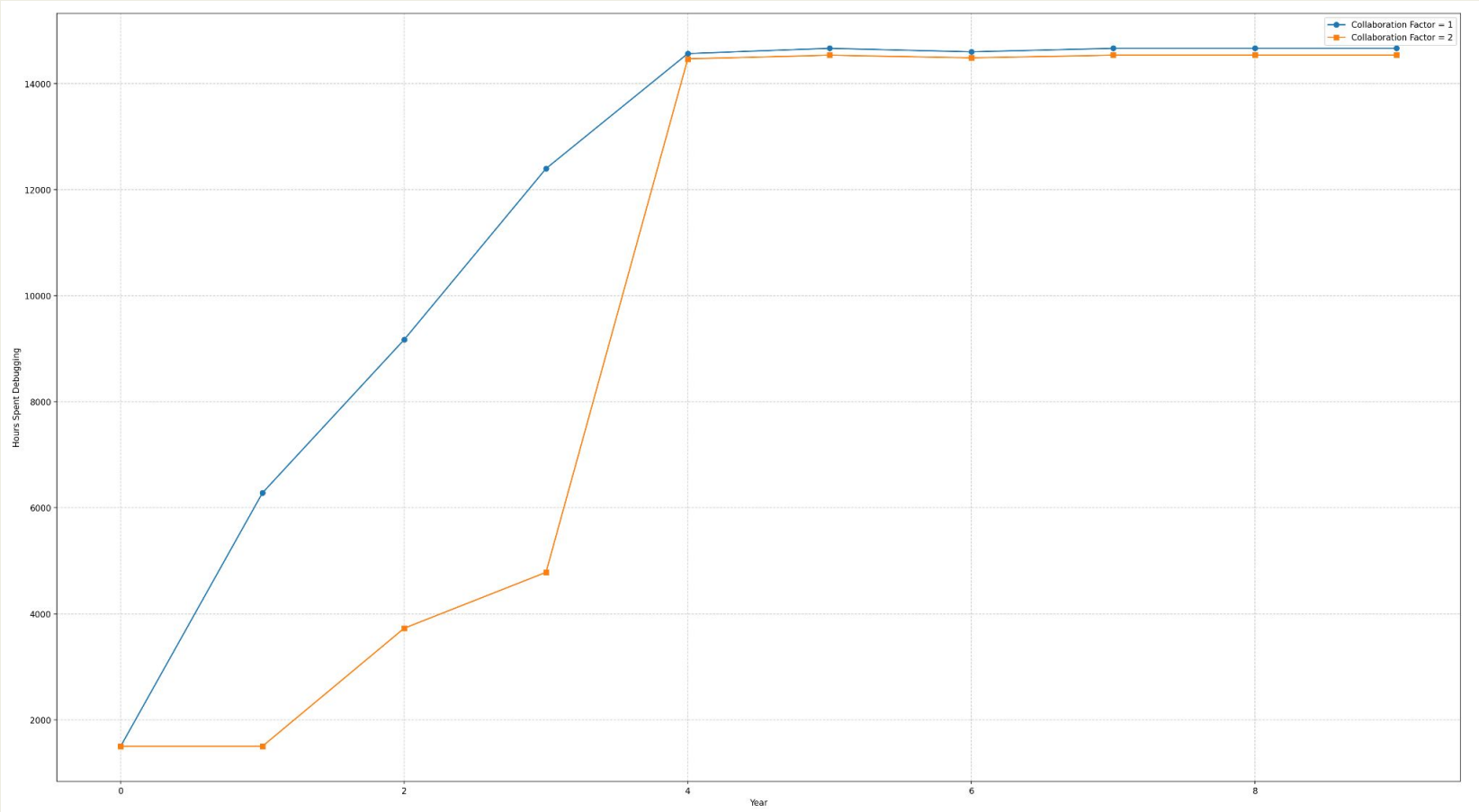
- Tautological Answer: Retain your employees
  - Retaining employees is a super power

- But it’s not always possible
  - Build robustness to this impact
  - This is also a super power

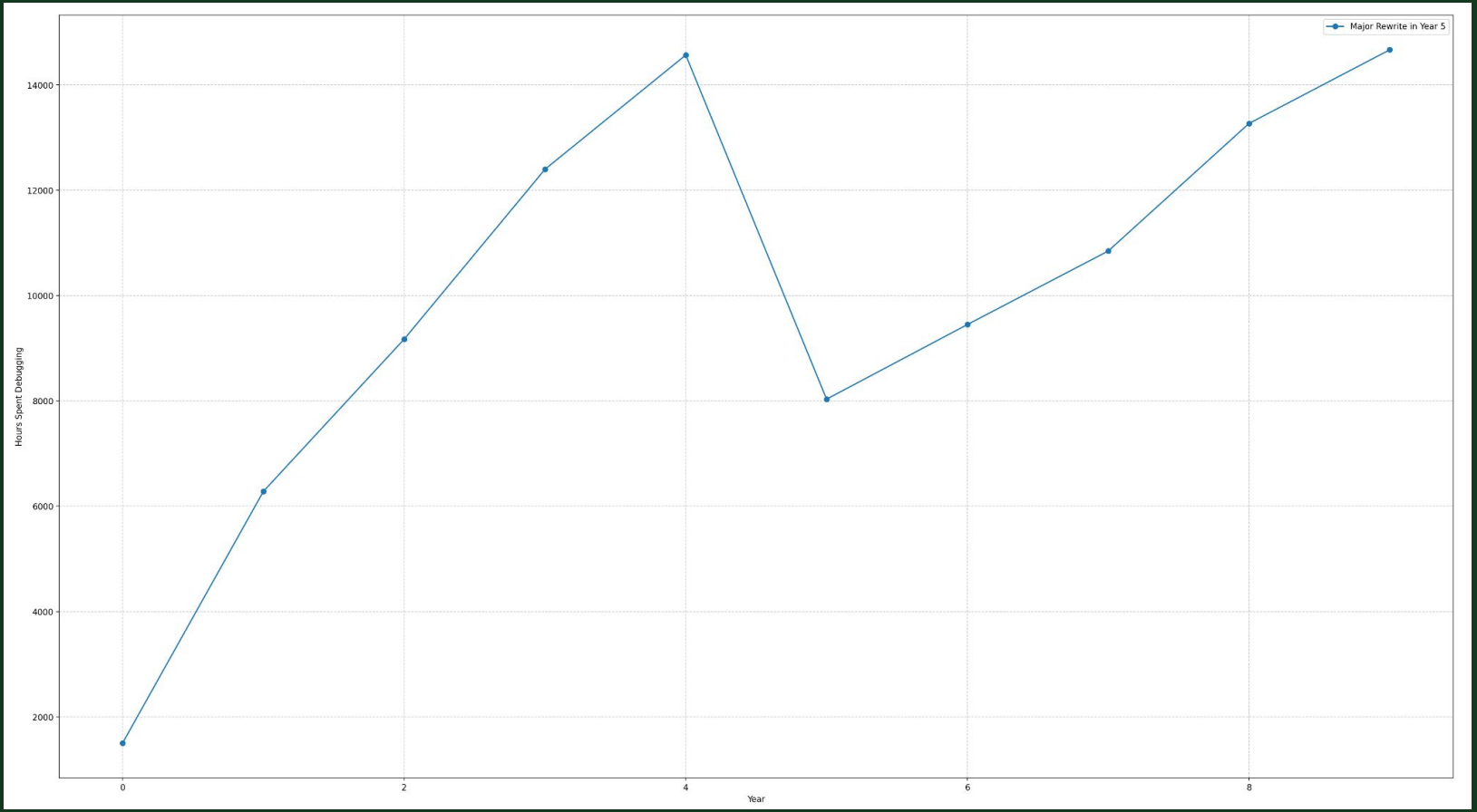
Implicit Assumption: Each LoC owned by 1 person



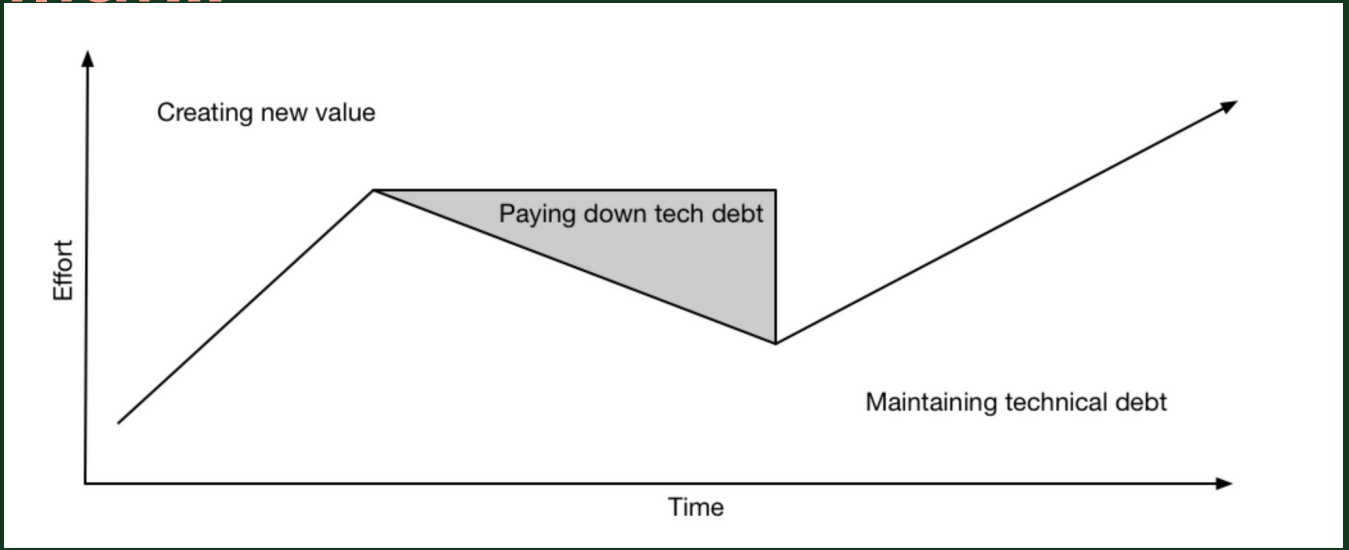
Mobbing, Pairing, and Sharing significantly improves impact



Let’s Migrate Half the Codebase!



“Hey that looks familiar...”



https://lethain.com/migrations/

“Migrations: The sole scalable fix to tech debt”

Migration is a set of muscles you need to build

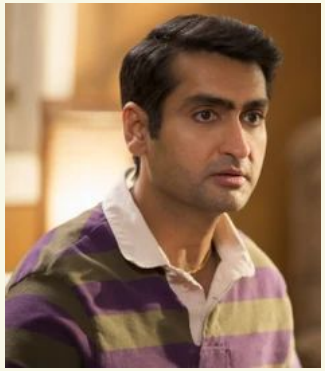
- Identify good targets for migration
- Recollect requirements
- Drive technical change, build consensus
- Derisk, Derisk, Derisk!
- Sell the change to the business

Beware

You-Touch-It-You

-Own-It-ism

If you see folks avoiding parts of the system that they don't understand, fix those incentives



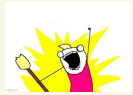
Engineering Managers

- Set up lunch and learns
- Reward digging into neglected parts of the system and bringing back knowledge

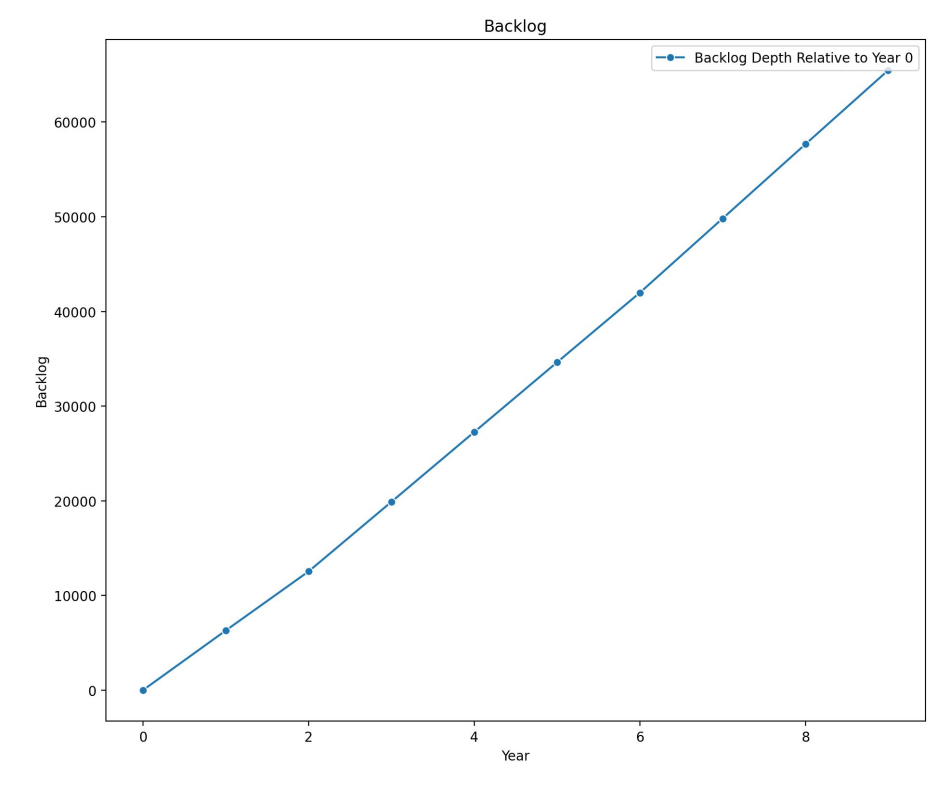
Principal/Staff Engineers

- Dive in and bring someone more junior with you, documenting everything
- Look for opportunities to rewrite/migrate

How to Fight Knowledge Decay

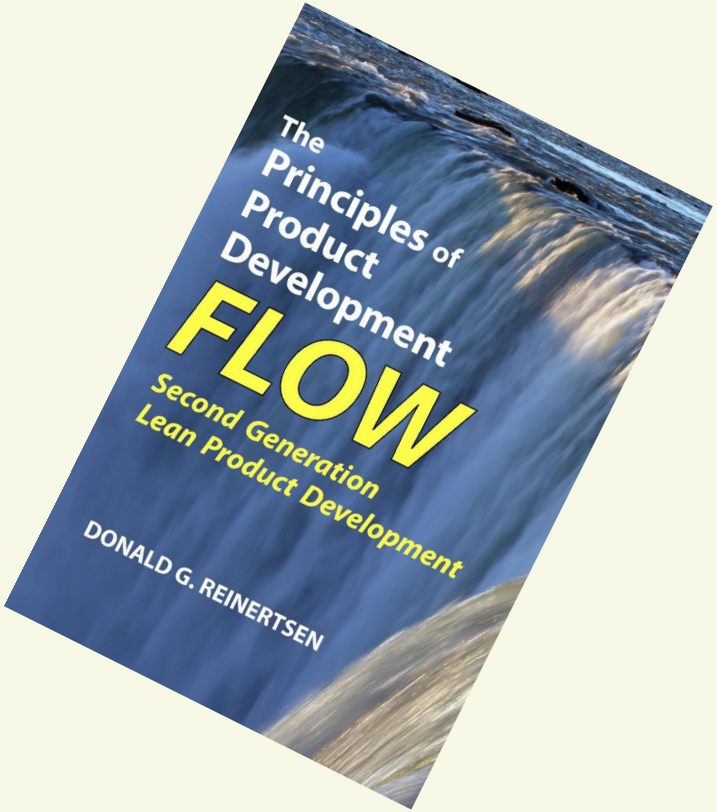
01	Get better at employee retention
02	Mob, Pair, and Share
03	Migrate All The Things 
04	Fight Copenhagen Culpability

Wayfinding


Backlog
Backlog Depth Relative to Year 0
60000
50000
40000
30000
20000
10000
0
Backlog
0
2
4
6
8
Year

“Principles of Product Development Flow”, Don Reinertsen

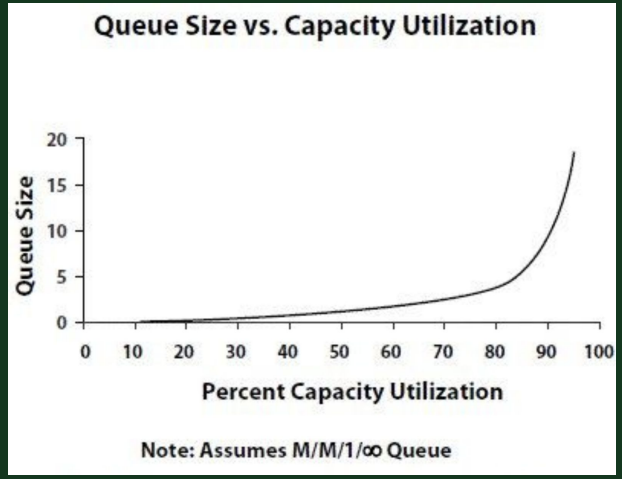
- Don’t saturate capacity
- Batch size matters

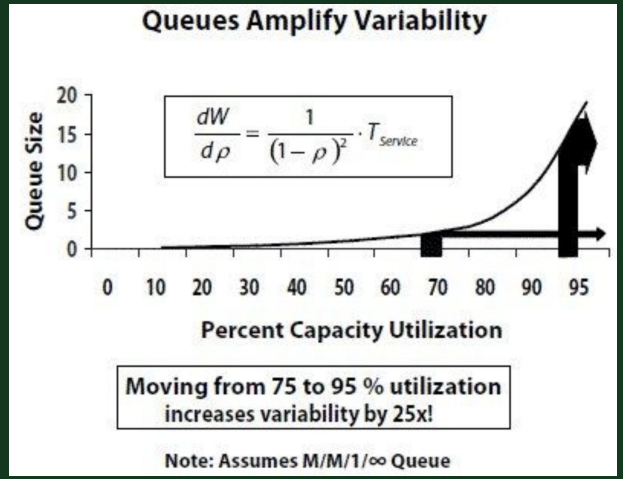


Once you get above ~75% utilization of a queue (DevOps team) you will see a massive drop in throughput.

Worse yet, your delivery will become much less predictable as work takes longer to complete.

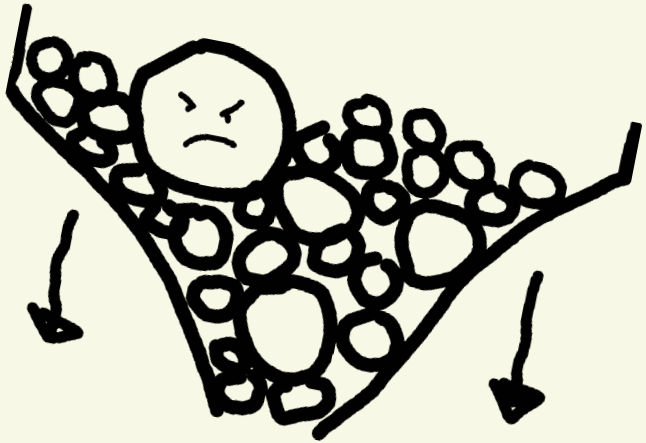
If your team has a significant amount of interrupt-driven work, you shouldn't plan additional work to take you over 75%





Large Batches of Work 😟 😫 💔

- Increases cycle time
- Increases variability
- Delays feedback
- Batch size has an exponential relationship with delivery time
- Self-reinforcing: Creates Large-batch dependencies elsewhere



How to Cutesy your Queues

01	Get rid of the aspirational backlog
02	Carve Up the Big Tasks
03	Swarm on long-tail work
04	Set WIP limits

Thanks for listening!

Extra Slides

Because I

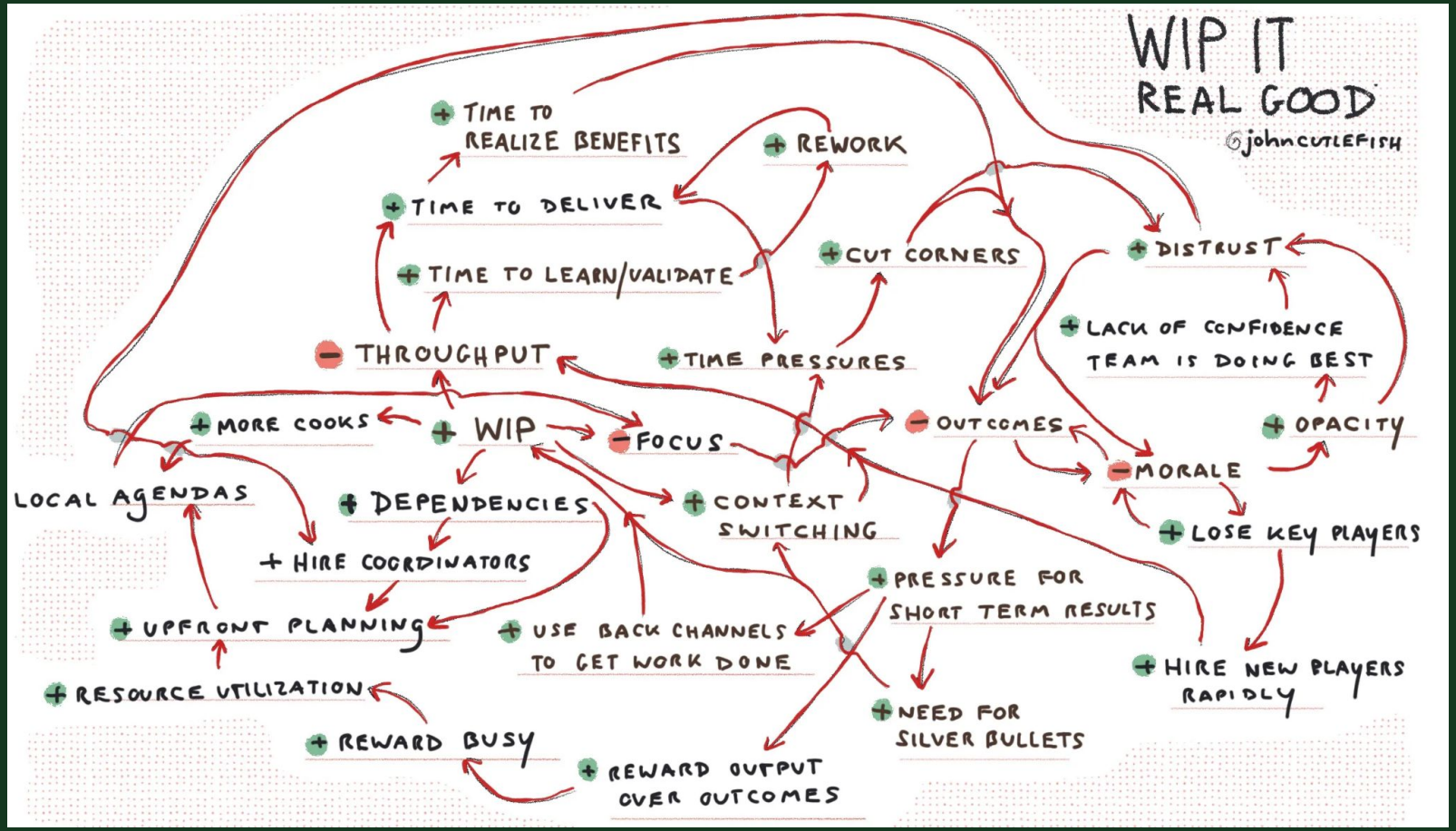
Talked Fast

WIP Limits

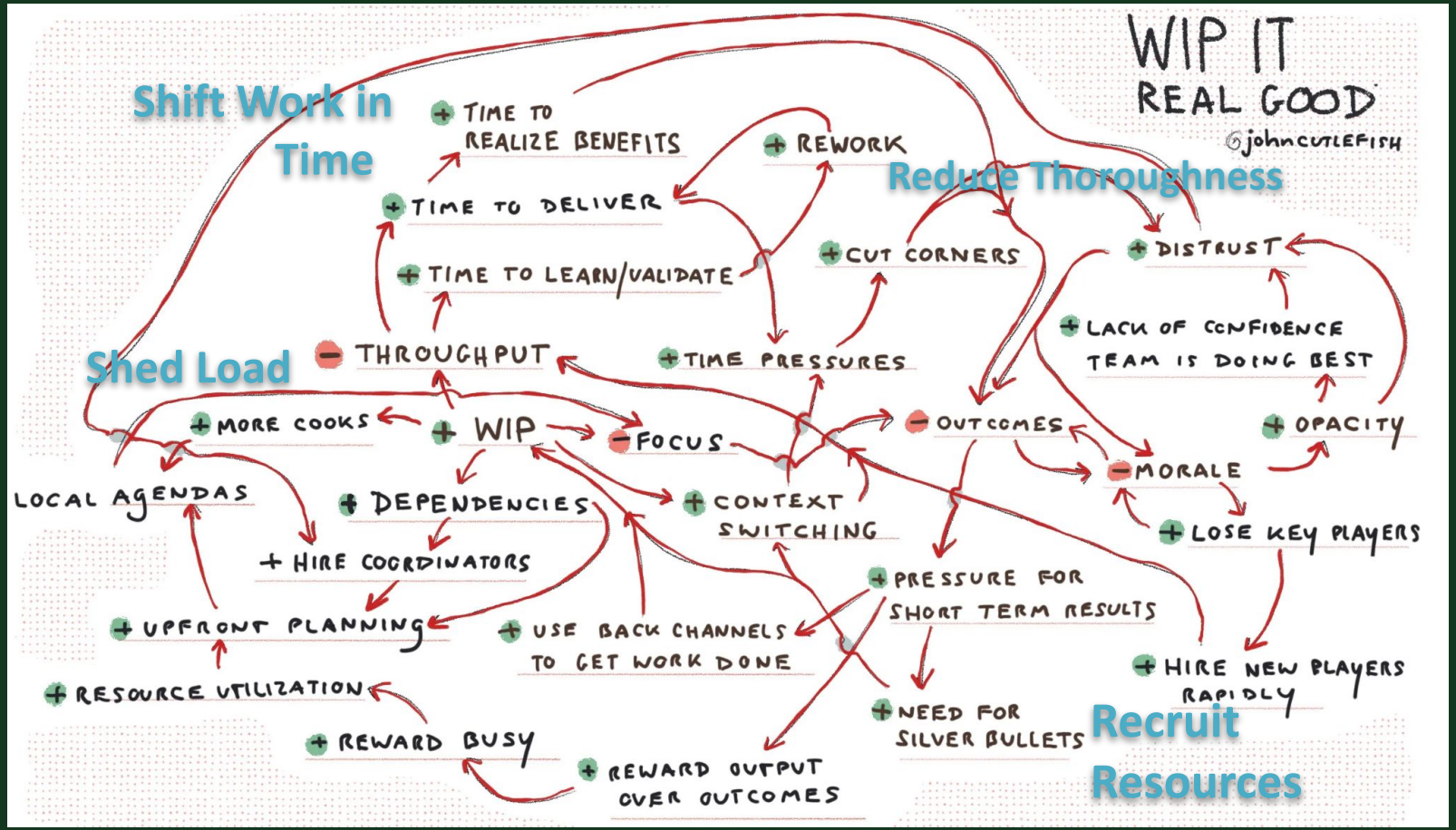
There are a lot of different philosophies on this

- Reinertsen: WIP limit twice your average WIP level
- Goldratt: Set the limit to the flow rate of biggest bottleneck
- Utilization Trap: Fewer than the number of servers
- Agile: Change, Measure, and Evaluate

I’m not a die hard believer in any one way, but all these folks agree that WIP limits improve flow rate and reduce variability.



https://medium.com/hackernoon/wip-it-real-good-66aa710178fd


WIP IT
REAL GOOD
©johncutlefish
Shift Work in Time
Reduce Thoroughness
Shed Load
Recruit Resources
+ TIME TO REALIZE BENEFITS
+ REWORK
+ TIME TO DELIVER
+ TIME TO LEARN/VALIDATE
+ CUT CORNERS
+ DISTRUST
+ LACK OF CONFIDENCE
TEAM IS DOING BEST
- THROUGHPUT
+ TIME PRESSURES
+ MORE COOKS
+ WIP
- FOCUS
- OUTCOMES
+ OPACITY
- MORALE
LOCAL AGENDAS
+ DEPENDENCIES
+ CONTEXT SWITCHING
+ LOSE KEY PLAYERS
+ HIRE COORDINATORS
+ PRESSURE FOR SHORT TERM RESULTS
+ UPFRONT PLANNING
+ USE BACK CHANNELS TO GET WORK DONE
+ HIRE NEW PLAYERS RAPIDLY
+ RESOURCE UTILIZATION
+ NEED FOR SILVER BULLETS
+ REWARD BUSY
+ REWARD OUTPUT OVER OUTCOMES